Using Some Database Principles to Improve Cooperation in Multi-Application Smart Cards

Sébastien Jean
Université de Lille, LIFL/RD2P
Cité scientifique, M3/111,
59655 Villeneuve d'Ascq, France
jean@lifl.fr

Didier Donsez
Université de Valenciennes,LAMIH/ROI
Le Mont Houy, BP 311,
59304 Valenciennes Cedex, France
didier.donsez@univ-valenciennes.fr

Sylvain Lecomte
Université de Valenciennes,LAMIH/ROI
Le Mont Houy, BP 311,
59304 Valenciennes Cedex, France
sylvain.lecomte@univ-valenciennes.fr

Abstract

Smart cards evolve today at the rate of non-embedded computing. If twenty years were necessary to integrate into smart cards the Database principles, it looks less than five years to embed Java. Emerging applications, such as loyalty ones, induce changes in smart card concepts. These applications are characterized by the gathering of several partners who bring each one their application part in order to offer global services to mobile users. Smart cards, which have been used as data servers for a long time, become now able to embed and run several collaborative applications. These applications lead to new requirements in embedded software lifecycle. It is indeed necessary for each partner to update his part without breaking the whole application. Moreover, it is necessary to allow the evolution of the cooperation scheme (modifications of resource sharing or arrival of new partners). Current smart cards models, databases or multi-applications cards, can not take these new needs fully into account. In this article, we show how the database principles, coupled to open smart card models, can however help to reach that goal.

1. Introduction

A smart card resides in the combination of a plastic support, contacts allowing the card to be aware of the outside world and a small component embedded in it. This monolithic component includes a microprocessor, that can be based on 8 bits CISC up to 32 bits RISC [4] architectures. As the size of the chip is constraint, there is only a limited amount of memory. This memory is divided into RAM (actually about 2 kilobytes) and non volatile memory (up to 64 kilobytes) such as EEPROM[1], or FlashRAM. Memory characteristics, like read/write time and granularity, have to be taken into account while evaluating performance. All the smart card concepts, from physical characteristics to file system organization, are normalised by [9, 10, 11, 13]. Smart cards are mainly dedicated to control access to informations they hold. This is done using hardware, operating system and application level mechanisms. The smart card communication protocol is a client/server one. The terminal requests the slotted card using APDU[2] commands.

For many reasons like security, mobility management or portability, smart cards are used in various and numerous application fields such as banking, healthcare, mobile phone, pay-TV, games, loyalty, etc. Nevertheless, these applications can be classified in three main groups:

- Portable and Personal Datafiles: Most common portable datafiles include health card, student card or transportation card. Personal data are stored and organized in directories and files or in relational tables. Data storage and manipulation are performed in a secure environment providing authentication and privacy.
- Access Control: These applications use smart cards as secure elements for identification process. The SIM[3]

[1] Electrically Erasable Programmable Read Only Memory
[2] Application Protocol Data Unit
[3] Subscriber Identification Module

card, adopted by the GSM[4] standards, the WIM[5] card for the WAP[6], or Pay TV are some good examples of access controlers.

- Payment: In this kind of applications, smart cards are used to secure electronic payments. Prepaid cards such as telephone ones are the simpliest kind of smart cards. Payment cards also include post-payment cards such as credit cards.

Since their invention thirty years ago, smart cards have become one of the most secured nomadic computing objects. Smart cards have always tried to integrate, faster and faster, the major concepts of *classical* operating systems (downloadable code, virtual machine, sandboxing, naming, ...) and information systems (databases, transactions, ...). Thus, if the Relational Databases principles [7] have been integrated only in 1992, Java has been embedded in a smart card in less than five years [8] after the first Java virtual machine announcement [16]. Nevertheless, these two previous concepts have always been considered separately. The smart card was indeed used either as a data server (for personal data, such as medical data, ...) or as an application server (for secure electronic payments, identification applications, ...). Emerging application models lead us to reconsider this separation. These new applications require more co-operation between each others and the capacity to evolve during the smart card lifecycle. The two previous smart card models (data server and application server) do not completely satisfy these needs.

In this paper, we intend to explain how the association of both models can reach this goal. We propose to use database smart cards mechanisms in order to get data independence in open smart cards. In Section 2, we present a state-of-the-art of the two major smart card execution models: database smart card and open smart card. Section 3 discusses about emerging smart card applications, their requirements and why the current smart cards models cannot fully support them. We propose, in Section 4, a new model, we called hybrid smart card, which intent is to take the benefits of both open smart card and database smart card models. Before concluding and presenting perspectives, Section 5 argues about implementation approaches and prototyping.

2. Database and Multi-applications Smart Cards

Advanced smart cards are used to develop complex and non-standard applications. Currently, smart card manufacturers offer to application developers the choice between two execution models. The first one is the database smart card which can store securely portable structured data files. The second model is the open smart card, which can be used to install, run and drop several applications written in a general-purpose language. In the next subsections, we describe these two smart cards execution models.

2.1. Database smart cards

Portable data files allow the gathering of data related to their owner on a mobile entity. As this data may be handled or shared by several users, mechanisms have to be provided in order to bring security. Database smart cards, where database principles are applied to store and manage data in a smart card, provide the most efficient way to build such portable data files. Furthermore, a database smart card can be seen as a mobile database as well as an element of a distributed database [7]. In *ISO 7816-7*-based database smart

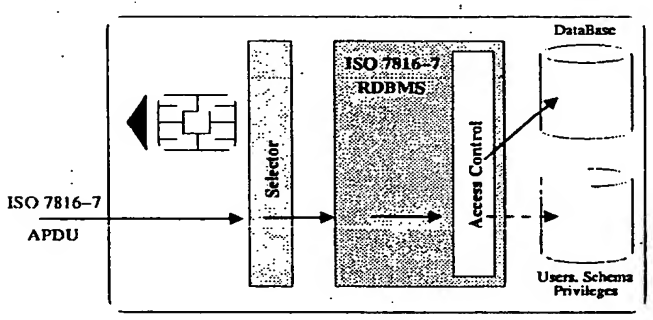

Figure 1. Database Smart Card Execution Model

cards [14], which execution model is presented on Figure 1, data are stored in a set of relational tables. The smart card operating system, which is based on an internal RDBMS[7] engine, provides a query language known as SCQL[8] (subset of SQL[9] [12]) to manage the data (definition, manipulation and control). Due to smart cards limitations, some concepts of relational databases, like join, are not available. In opposition to *file system* smart cards that handle data through files, database smart cards operating systems provide fine-grain access control (user, table, and view). Every user is provided (by the owner of corresponding objects) with a set of access rights (SELECT, INSERT, DELETE, and UPDATE) that allows him to send a set of queries in order to handle the whole data set or just a subset. Database smart cards, as the others kinds of smart cards, are seen as servers. When receiving a query (that has been forwarded by the

[4] Global System for Mobile communications
[5] WAP Identification Module
[6] Wireless Application Protocol
[7] Relational DataBase Management System
[8] Structured Card Query Language
[9] Structured Query Language

APDU Selector), the database engine computes it and eventually sends back results. Since database smart cards do not have advanced application facilities, like stored procedures, applications take place on the host system (*i.e.* where the smart card is plugged).

2.2. Multi-applications smart cards

Multi-applications smart cards (also called open smart cards), whose execution model is presented on the Figure 2, are characterized by three main concepts: execution of several applications (called either applets or card applets) within the same card, dynamic application loading/withdrawal, and cooperation between applications.

Multi-applications smart cards, like Windows for Smart Card, MULTOS [17] or Java Card [5], break the old smart card software monolith model because the application is no longer associated with the card operating system. Then, as the content becomes extensible, several applications developed by different providers can be embedded within the same card [21] and, consequently, software protection mechanisms have to be provided to ensure application security. The panel of services that can be offered to a user is more and more vast. However, because of the technical limits of the smart card, just a little subset of this panel can be stored in a card at a time. Thus, multi-applications smart cards provide flexibility since services can be added or removed during the smart card lifecycle. Then, a user can decide to add a service he needs and to withdraw from another when it becomes useless. Nevertheless, the application downloading has to be highly secured, particularly if on-line downloading (through GSM or Internet for example) is allowed. Nevertheless, the coexistence of several applications takes another dimension if these ones can interoperate (sharing information either by data or by code).

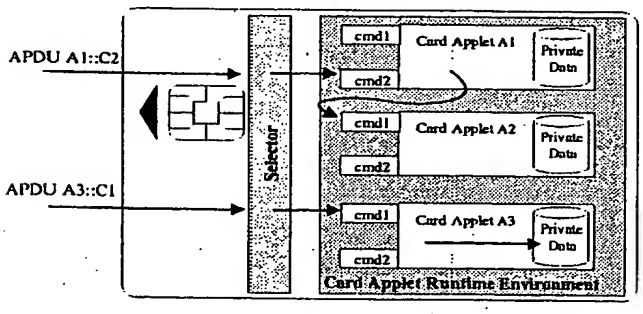

Figure 2. Open Smart Card Execution Model (example of Java Card)

3. New Applications Requirements Vs Smart Cards Limitations

In this Section, we focus on the emerging case of multi-partners service providing in open smart cards. We firstly present what the smart card lifecycle looks like, underlying the requirements of cooperation, security and upgrading facility. Then, we show where the current smart card models do not satisfy all these needs.

3.1. What smart cards need while opening up to service providers

Nowadays, open smart cards applications trends go to cooperative service providing, as smart cards open to mass market and non-specialist application designers. Loyalty points management is an example of that kind of new applications where each provider involved brings its component (*i.e.* how the points are managed, ...) to build the entire application. Gambling is another example of combined applications that seamlessly offer a service to end-users. An e-purse application can also be combined with the loyalty one to enforce the consumer/merchants relationships. This cooperation, that can be realized by using either code or data sharing, has to be secured to ensure that the information sharing scheme is respected. So, this need of secure cooperation is the first of the requirements raised by this novel scheme. The second need of this new kind of application is the flexibility of evolution. This consists of two main steps: the application update and the evolution of the cooperation scheme. First, embedded services and data have to be easily updated. This particularly means that the card holder should not have to send back the card to its issuer (who is usually one of the service providers). Actually, multi-applications smart cards do not assume such evolution. The evolution flexibility of applications is ensured if each service provider can update its contribution without breaking the whole application. Second, the evolution of the cooperation scheme has to be also flexible and dynamic. It must be possible, whenever during the smart card lifecycle, to change rights on objects (data or code) or to take into account the arrival or the withdrawal of partners. In the case of a cooperation realized through data sharing, it requires data independence as well as structured data. In both cases, dynamic rights management is necessary.

3.2. Current smart cards limitations

The previously outlined requirements for multi-partners applications are secure cooperation and flexible upgrading. Each of the two main smart cards models, Database and open smart card, has advantages and drawbacks while being used in such a scenario. Database smart cards provide

dynamic schema evolution and fine-grain access control. These mechanisms satisfy two of the previous constraints, which are data sharing security and evolution flexibility. In addition, these smart cards offer both structured data and powerful data manipulation tools. In contrast, their main drawback is that there is no way to execute embedded applications. Some researches have nevertheless studied the application of active databases concepts in smart card area [19]. As previously said, the application is hosted by the terminal, which sequentially sends data manipulation orders. Due to this lack of application embedding facility, databases smart cards are enclosed in a tiny application range, which is the one of portable data files. Advantages of multi-applications cards are obvious. They provide an execution platform for several embedded services, which breaks the smart card memory constraint. Nevertheless, data are not globally structured or manipulated. Each application has to manage its shareable data, using encapsulated data manipulated via interfaces (like Java Card does) or sharing data through files (like Windows for Smart Card does). The major drawback is then the cooperation evolution, mostly in the case where the owner of shared data changes the way to manage it. Open smart cards do not provide also fine-grain rights management. Due to that, one often has to choose between security and more cooperation.

4. The Hybrid Smart Card: a Federating Approach

The recent arrival of open smart cards marked a turning point in the evolution of the card business model. It becomes possible to embedded applets provided by several organisations within a card. However, when a card is shared by several partners, partners' applets may cooperate within the card by sharing information. Such multi-partners applications have been previously presented in Section 3. These emerging applications require secure cooperation and flexibility of evolution. Our motivations, when defining the hybrid smart card model, are to satisfy these needs. The Hybrid smart card gathers both the functionalities of database and open smart cards. The providing of a database API for open smart cards has already been studied in [3], but the innovative idea in which is based the Hybrid smart card model, is to use the database concepts as a global support for data and code co-operation between the partners involved in a smart card application. In this Section, we firstly present the Hybrid smart card execution model. Then, we discuss about the extension of access control and about the installation of applications.

4.1. Hybrid Smart Card Execution Model

The Hybrid smart card can be seen according to three external facets presented on the Figure 3. In the *Database mode*, the user (local or remote) sees the smart card as a database one. He can send SCQL APDUs that are processed by the internal DB engine. In the *Multi-applications mode*, the APDUs sent are requests to services (example of the APDU A1::C1) that use only private persistent data (stored in the isolated space of the application). This case is the one of one-partner services without cooperation. Finally, in the *Hybrid mode*, the APDUs are still services calls (example of APDUs A2::C1 or A3::C2); but these multi-partners services are designed to cooperate through the database mechanisms. These services can handle the database objects directly or through a cooperation scheme.

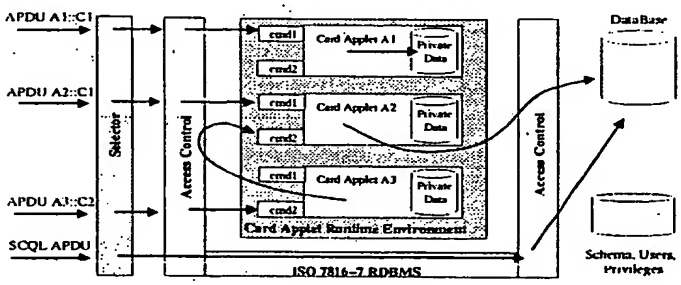

Figure 3. Hybrid Smart Card Execution Model

4.2. Extending the access control to applications

In the hybrid model, the internal database engine manages a list of users. It is able to authenticate these users and to check their privileges on database objects. If we consider cooperation through data, the use of the internal RDBMS allows satisfying the need of flexibility. Indeed, the data independence and the dynamic evolution of the schema provide a flexible way to manage information sharing between several partners.

Since we intend to provide also cooperation through code, we propose to extend the database access control mechanism to the embedded applications. An application is thus considered as a PACKAGE, which consists of a set of stored procedures and shared data. We add some items to the privileges list in order to take into account this extension: CREATE PACKAGE, DROP PACKAGE and EXECUTE PROCEDURE. Since the manipulation of applications can be done in two different ways (*i.e.* through the *multi-applications* and the *database* facets), we couple these new privileges to the matching multi-applications

mechanisms (adding or withdrawal of card applets, method invocation). This approach is similar to the Java Stored Procedure used by Oracle (in the version 8i) [1] to execute stored procedures (written in Java) on the database server. The data access security is transposed to applications since the EXECUTE PROCEDURE privilege is checked at runtime by the method itself (by giving user's identity to the RDBMS). Extending the use of the SCQL engine to applications allows reaching the objectives of cooperation security and of evolution flexibility (for cooperation as well as for services update).

The database smart cards provide an access control mechanism that works only with data (TABLE). In addition, in the multi-applications cards, the application installation process is done according to an established protocol. For these reasons, we have chosen to allow an application installer to optionally set some privileges on application methods. The installer specifies, by the way of a SCQL APDU, the privileges associated to the users that he allows to access to these PROCEDURE objects. These privileges are stored in a system table, and a procedure is considered as PUBLIC if there is no matching entry in the associated system table. A PUBLIC procedure can be invoked by every anonymous or identified user. Since the procedures of an installed application are considered as PUBLIC by default (till privileges are set), the installation and privilege setting have to be atomic (to prevent attacks based on smart card tear). Transactional mechanisms have already been studied in the case of smart cards [18]. A secure solution is to set privileges before the application installation. If the card is removed, this solution can induce inconsistency but it does not affect the checking of well-set privileges.

5. Implementation of the Hybrid Smart Card

This Section firstly discusses about the various possible approaches for the implementation of the hybrid smart card. Finally, we present protyping issues and evaluation.

5.1. A unique tool for two different approaches

In order to realize an implementation of the hybrid smart card model, two kinds of approaches can be considered. The first way is to rebuild from scratch a smart card operating system that runs natively both an open smart card virtual machine (like the Java Card Runtime Environment) and a RDBMS engine. Because building an efficient open smart card virtual machine an efficient RDBMS is a lot of tayloring, tuning and assembly work, we have not yet considered it for prototyping. Nevertheless, we will discuss the advantages of such an approach while presenting the prototyping results in the next Subsection. This can also be done on top of an open smart card operating system, like the JavaCard or the WFSC one. The advantage of such a choice is its compliance with widespread standards and products. In the case of the Java Card, it resides in the realization of an applet that implements the internal RDBMS. This applet must accept the complete SCQL orders set to be compliant to the previously presented database mode. The hybrid mode can then be provided by using the SIO[10] mechanism of the Java Card. This mechanism allows a card applet to share a part of its functionalities with one or more others. This applet must only implement a subclass of the Shareable Interface, that defines what methods or attributes are visible. Using SIO, the hybrid mode is supported by enabling the cardlets to use the internal database (in a cooperative way or not). The interface of the DB applet can be a SQL/CLI-like API, or an API closer to SCQL orders. In the first case, with APIs like the JDBC-SC [3] one that looks like JDBC[11] [22], the statement is a non-interpreted character string. The RDBMS engine analyzes the string, computes the query with local tables and returns a result set of matching lines. In traditional Client-Server programming, this approach makes applications independent from the data source format or from the native API of the RDBMS server, but it needs sophisticated interpreter on the server. Since sophisticated sounds large footprint, an SQL/CLI API cannot be implemented as is in a smart card to interface card applications with the database engine. In order to avoid the use of such an analyser, the set of queries has to be split into a set of dedicated methods like executeSelect(...), executeUpdate(...) and the attributes have to be manipulated through row offsets instead of strings. In the second case, each SCQL order must be wrapped into a method call (e.g. the CREATEUSER order is wrapped into DB.createUser(byte[]name)).

We propose to use the well-known SQL embedding technique to provide a tool that eases application design and that makes it independent of the target APIs, like those previously given or others like JDBC-SC, ... Embedding high level SCQL statements in the application source also alleviates the burden of theirs translations. To handle embedded SCQL statements, we use a pre-compiler that statically:

- analyses each SCQL statement,
- checks the naming and the typing of tables and columns,
- translates SCQL statements according to the underlying API,
- and finally inserts SCQL wrapped statements in the application source.

The pre-compiler inserts also an access control statement at the beginning of each application command in order to

[10] Shareable Interface Object
[11] Java DataBase Connectivity

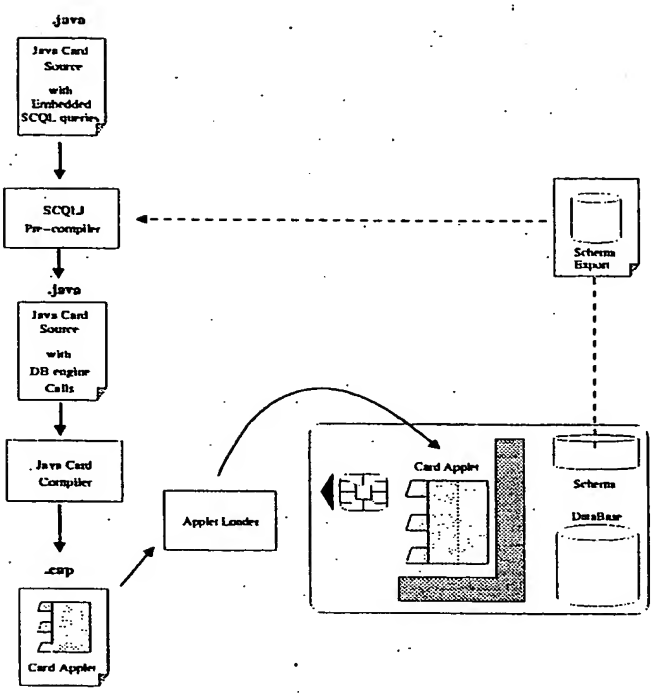

Figure 4. Java Card Applets Generation for Hybrid Smart Card Model

respect the cooperation scheme security. Another feature of this approach is that the precompiler is able to check the existence of tables and views as well as types matching before application embedding. This can be done since the database schema can be exported from the smart card. Thus, some errors can be avoided statically before installing the application. The export of the database schema is also an help for the application designer because he can take into account its evolutions when he wants to update his application part. Since access control orders are inserted at the beginning of each method by the precompiler, the code becomes lighter in the case of PUBLIC methods (this has to be point out when remembering the smart cards memory constraints). In the case of the Java Card, the resulting file is a Java source file that can be compiled, loaded and installed in the card. The corresponding production steps are described by the Figure 4. This precompiler named SCQLJ uses a subset of the SQLJ syntax to embed SCQL in the application source. The SCQLJ pre-compiler converts embedded SQL statements nested in Java programs into JDBC-like calls.

5.2. Prototype

The prototype of the Hybrid Smart Card, is based on a Java Card (compliant with the 2.1 specification version [20]) in which we have embedded a shareable applet (as presented in the previous Subsection) that runs a RDBMS engine. Test processing is done using OpenCard Framework which helps designing client applications. The prototype includes both the JDBC-like API (called JCDB) and the SCQLJ precompiler which translates embedded SCQL statements into JCDB methods calls. The SCQL-compliant database mode has also been implemented. The DB applet resides in four main parts. The first is the applet interface defines all the operations availiable for the DB manipulation by the others cardlets. This operation set includes connection management (open/close for anonymous or authenticated users), data definition operations (users, tables, columns) and data manipulation operations (select, insert, update, ...). The second is the core API class that implements the RDBMS engine. The third is a JDBC-like API that provide objects like Statement, ResultSet, Connection. Finally, the fourth part is a set of classes that defines meta-objects (Table, User, Column, View). The SCQLJ precompiler is partly implemented with JavaLex and B/Yacc, and provides static database schema and types checking.

The implementation is nearly finished. The size of the whole package is about 20 kilobytes, including the space used by the memory manager (that stores rows in a raw array of 4 kilobytes, organized as a set of size-fixed chunks without defragmentation). Unfortunately, some features still remain unchecked because of the current Java Card products (existing simulators, when prototyping, were unsatisfying because based on Java instead of Java Card language). Our prototyping platform was providing only 15 kilobytes for applets storage. The first experiments on the prototype allow us however to validate the hybrid smart card model. Even if it fewly overruns memory availiability, this should not remain a problem according to technical evolutions of smart cards. he overhead induced by access control when invoking applets methods is tiny, and Database management is not prohibitive since delays are reasonable.

However, this implementation choice has some drawbacks. First, the Shareable Interface Object mechanism of the Java Card is not really adequate, easy to use or efficient. The necessary static allocation of data storage zone in the non volatile memory (non transient) is also not efficient, but this is due to Java Card object allocation model and the lack of garbage collection. More than a simple oversizing problem, the lack of performance comes from the crossing all the software layers since the database physical memory must be managed at the application level rather than at the operating system level. Then, the memory manager rely in this case on an unefficient manipulation of byte arrays provided by the Java Card virtual machine. For that reason, it is much more interesting to build another smart card operating system, as previously said, in order to place the RDBMS engine at the operating system level. In this way,

small footprints and efficient DB engines that match much more smart cards constraints, such as the picoDBMS [2] one, can be realized. Another benefit from building another smart card operating system is the possible implementation of an efficient persistent recoverable memory, which is not yet implemented in the existing ones. This kind of memory management has already been studied and prototyped on the CASCADE platform [6].

6. Conclusion

In this paper, we have shown how open smart cards can take benefits of the database principles in order to offer an efficient support for cooperation between embedded applications. Emerging applications, which involve several partners that want to provide a global service to their nomadic subscribers, induce new requirements in terms of cooperation and evolution. The federated model we propose, the Hybrid smart card, is based on the Database principles. In our approach, the multi-applications part is coupled with the database one by the way of an extension to the applications of both sharing and access control facilities. Even if the prototype (based on Java Card) has validated this approach, its efficiency is limited by several constraints inherent to the implementation choices. A better way is to implement from scratch another smart card operating system that could include low level mechanisms such as recoverable memory management.

Nevertheless, the multi-applications smart card use has nowadays sense only because they can be integrated into large scale information systems. Besides an internal interoperability between embedded applications, an external interoperability occurs between these applications and remote services. This external interoperability is actually one-way since the embedded applications cannot invoke remote services. Nevertheless, some recent studies show the benefits of complete smart card interoperability [15] (*i.e.* when smart cards can react by sending as well orders to the outside world). However, one must remember that the main characteristic of a smart card is its intrinsic security. If embedded applications can engage a dialog with external hosts, the same security level has to be ensured in order to protect embedded data or code. We actually study an extension of this Hybrid smart card model in order to take into account this new requirement. This one is based on the integration of external resources as objects of the embedded database.

References

[1] SQLJ: Embedded SQL in Java. Oracle White paper, 1999.

[2] C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez. PicoDMBS: Scaling Down Database 'Techniques for the Smart Card. In 26th *International Conference on Very Large Data Bases (VLDB 2000)*, pages 11–20, 2000.

[3] L. Carrasco. RDBMS's for Java Cards ? What a senseless idea ! ISOL Corp. White paper, 1999.

[4] CASCADE : Chip Architecture for SmartCArd and intelligent DEvices, European ESPRIT project (EP8670).

[5] Z. Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison-Wesley, June 2000.

[6] D. Donsez, G. Grimaud, and S. Lecomte. Recoverable Persistant Memory for Smart Card. In J.-J. Quisquater and B. Schneier Eds, 3rd *Smart Card Research and Advanced Application Conference (CARDIS'98)*, LNCS, Springer-Verlag, 1998.

[7] G. Grimonprez and E. Gordons. A Card as Element of a Distributed Database. In *IFIP WG 8.4 Workshop*, Ottawa, Canada, 1992.

[8] S. Guthery. Java Card: Internet Computing on a Smart Card. *IEEE Internet Computing*, 1(1):57–59, 1997.

[9] International Standard Organisation. Information Technology - Identification Cards - Integrated Circuit(s) Cards with Contacts. Part 1 : Physical Characteristics. 1987.

[10] Internatjonal Standard Organisation. Information Technology - Identification Cards - Integrated Circuit(s) Cards with Contacts. Part 2 : Dimensions and Location of the Contacts. 1988.

[11] International Standard Organisation. Information Technology - Identification Cards - Integrated Circuit(s) Cards with Contacts. Part 3 : Electronic Signals and Transmission Protocols. 1989.

[12] International Standard Organisation. Information Technology - Database Languages - SQL, 1992.

[13] International Standard Organisation. Information Technology - Identification Cards - Integrated Circuit(s) Cards with Contacts. Part 4 : Inter-Industry Commands for Interchange. 1994.

[14] International Standard Organisation. Information Technology - Identification Cards - Integrated Circuit(s) Cards with Contacts. Part 7 : Inter-Industry Commands for Structured Card Query Language (SCQL). 1999.

[15] S. Jean, D. Donsez, and S. Lecomte. Smart Card Integration in Distributed Information Systems : the Interactive Execution Model. In 1st *IEEE International Symposium on Advanced Distributed Systems (ISADS'2000)*, 2000.

[16] D. Kramer. The Java Platform : A White Paper. Sun Microsystems White paper, 1996.

[17] J. Langavant. The MULTOS System and Architecture. In 1st *Gemplus Developer Conference(GDC'99)*, 1999.

[18] S. Lecomte, G. Grimaud, and D. Donsez. Implementation of Transactional Mechanisms for Open Smart Card. In 1st *Gemplus Developer Conference(GDC'99)*, 1999.

[19] B. Noclerc and J.-M. Place. An Advanced Card Operating System on the Cascade Chip. In *EMMSEC'97 Conf.*, 1997.

[20] Sun MicroSystems. Java Card 2.1.1 Runtime Environment (JCRE) Specification, 2000.

[21] J.-J. Vandewalle. Loading Several Services into Multi-Purpose Integrated Circuit Cards: Distribute Functions to Users, Gather Data into Cards ! In *European Research Seminar on Advances in Distributed Systems*, 1995.

[22] S. White, M. Fisher, R. Catell, G. Hamilton, and M. Hapner. *JDBC API and Reference*. Addison-Wesley, 1999.